Improvement of the FFT-based Poisson Solver in IMPACT

X. Sherry Li   Ji Qiang
Lawrence Berkeley National Lab

ComPASS Meeting, UCLA, Dec. 2-3, 2008

Outline

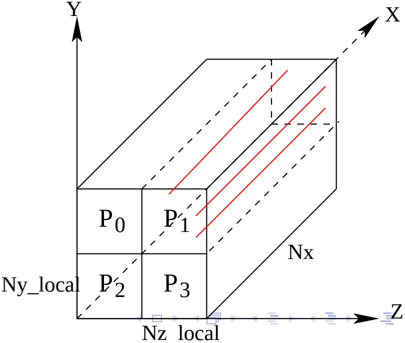
IMPACT-Z

- Model high intensity, high brightness beams in linear accelerators (Poisson-Vlasov integral equation)
- 3D Particle-In-Cell code with two domains: charged particles and the electric field generated by the charged particles
- Simulation cycle:
  1. deposit charge density on grid points
  2. solve Poisson equation for field vector (now FFT-based)
  3. interpolate the field vector
  4. advance particles
- 6 Poisson solvers for different Boundary Conditions:
  - transverse open or closed BC with round or rectangular shape
  - longitudinal open or periodic boundary conditions

IMPACT-Z Parallelization

Parallelization is based on Domain Decomposition:

- 3D grids $N_x \times N_y \times N_z$
- 2D processor mesh $P = P_y \times P_z$ is used for the block distribution on the y–z plane.
- Grid points along N_x dimension are local to one processor, and each processor holds a block column of the grid points along N_x dimension.



Case of open BC

- Method based on convolution of the Green function
  (1) forward FFT
  (2) forward FFT to compute convolution of the Green function
  (3) inverse FFT

  all with double-sized computational domain

Case of open BC

- Method based on convolution of the Green function
  1. forward FFT
  2. forward FFT to compute convolution of the Green function
  3. inverse FFT

  all with double-sized computational domain

- General steps of parallel 3D FFT
  - 1D FFT along X (local)
  - transpose (communication)
  - 1D FFT along Y (local)
  - transpose (communication)
  - 1D FFT along Z (local)
  - (optional) transpose (communication)

Open BC Poisson solver

(1) FFT3D

1. $fft1d(N_x, N_y^{local} * N_z^{local})$ **(real-to-complex)**
2. transpose3d$(y \rightarrow x)$
3. $fft1d(N_y, N_x^{local} * N_z^{local})$
4. transpose3d$(z \rightarrow x)$
5. $fft1d(N_z, N_x^{local} * N_y^{local})$

(2) GreenFunction

1. $fft1d(N_x, N_y^{local} * N_z^{local})$ **(real-to-complex)**
2. transpose3d$(y \rightarrow x)$
3. $fft1d(N_y, N_x^{local} * N_z^{local})$
4. transpose3d$(z \rightarrow x)$
5. $fft1d(N_z, N_x^{local} * N_y^{local})$

(3) INVFFT3D

1. $invfft1d(N_z, N_x^{local} * N_y^{local})$
2. transpose3d$(y \rightarrow z)$
3. $invfft1d(N_y, N_x^{local} * N_z^{local})$
4. transpose3d$(x \rightarrow z)$
5. $invfft1d(N_x, N_y^{local} * N_z^{local})$ **(complex-to-real)**

Code optimization

- Multiple 1D FFTs with same length.
  Each function $fft1d(n, m)$ involves a distributed 3D array of size (n, l_2, l_3), where $m = l_2 * l_3$.
  - **OLD:** wraps m loops around the call to each individual 1D FFTW function.
  - **NEW:** use the FFTW function that takes as input the multiple vectors of the same length, so that the plan is created once and reused m times.
- Real-complex mixed data transformations.
  - **OLD:** first converts real data to complex data, then calls a complex-complex transform.
  - **NEW:** directly calls the real-to-complex or complex-to-real functions (available in FFTW 2.1.5), saveing half of the operations.

Case of closed BC

- Example: rectangular pipe with transverse finite and longitudinal open
- Only involves Sine transform, which can be obtained by FFT
- 3D FFT structure in Poisson solver

1. transpose3d($y \rightarrow x$)
2. $sinft(N_y, N_x^{local} * N_z^{local})$ (**real-to-complex**)
3. transpose3d($x \rightarrow y$)

- Similar optimizations : exploit multiple transformations of the same length and real-complex mixed data types

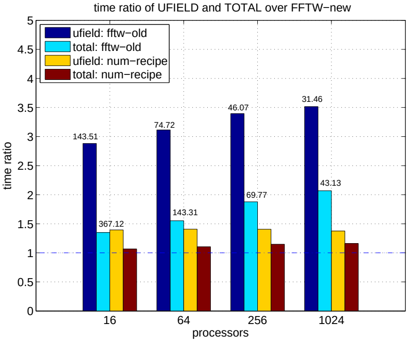
Benchmark configuration

- Codes
  - `fftw-new`
  - `fftw-old`
  - `num-recipe - Numerical Recipe`
    - (–) length limited by power-of-two; copyright issue
- Inputs
  - 20M particles on 128^3 grids ($\sim$ 10 particles per grid point)
  - 40M particles on 128^3 grids
- Machines

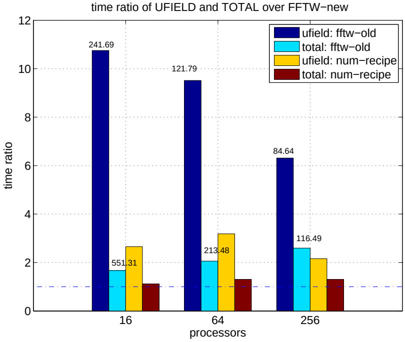
System	Cray XT4 (Opteron) (franklin)	IBM Power5 (575) (bassi)
Clock (GHz)	2.6	1.9
DP Gflops/Core	5.2	7.6
Cores/Node	2	8
OS	Compute Node Linux	AIX
Compiler	ftn -O3 -fastsse	mpxlf90_r -O3 -qstrict

Results – time-ratio of OLD over NEW

- Time breakdown: ufield (field solver), total



(a) Cray XT4



(b) IBM Power5

Summary of results

- Statistics of open BC case

	Cray XT4 (Opteron) (franklin)	IBM Power5 (575) (bassi)
Improvement		
ufield	3.5 x	10 x
total	2 x	2 x
Fraction of time in ufield		
`fftw-old`	39-72%	40-72%
`fftw-new`	18-42%	6-29%

Summary of results

- Statistics of open BC case

	Cray XT4 (Opteron) (franklin)	IBM Power5 (575) (bassi)
Improvement		
ufield	3.5 x	10 x
total	2 x	2 x
Fraction of time in ufield		
`fftw-old`	39-72%	40-72%
`fftw-new`	18-42%	6-29%

- Case of the closed BC
  - ufiled improved 32%, total improved 8%

Future work

- Poisson solver
  - possibility of improving transpose in 3D FFT
  - non-FFT based Poisson solver, such as multigrid-based, which has better algorithmic complexity
    - boundary conditions?
- Part of the code other than Poisson solver

QUESTIONS at the Meeting

- "plan creation" done only once, memorize it, then pass around an extra "plan" argument for all the relevant routines
- need BC other than those implemented in IMPACT?
- iterative solver starting from the result of the Poisson solver at a previous step